Introduction à la sécurité – Cours 8
Infrastructure de clés publiques

Catalin Dima

Gestion des clés

- ♦ La gestion des clés concerne :
  - – La distribution de clés cryptographiques,
  - – Les mécanismes utilisés pour l'association identité–clé
  - – La génération, la maintenance et la révocation de clés.
  - – Le stockage sécurisé à long terme des clés de déchiffrement (obligation légale en France).
- ♦ Problématique :
  - – Comment associer une clé publique à une (représentation d'une) identité ?
  - – Comment représenter une identité (nommage) ?
  - – Comment sécuriser la création des identités ?
  - – Comment limiter les dégâts provoqués par la perte d'une clé secrète ?
  - – Comment informer à temps tous les membres de la communauté de cette perte ?

“Public key infrastructure” (PKI)

- ⧫ Système permettant aux agents de reconnaître quelle clé publique appartient à qui.
- ⧫ *Services* fournis par la PKI :
  - ⋆ **Vérification d’identité et création de certificats**
  - ⋆ **Révocation des clés**
  - ⋆ **Test d’appartenence de certificats**
  - ⋆ **Recouvrement des clés de déchiffrement**
- ⧫ Composants d’une PKI :
  - **Authorité de certification** (CA).
  - **Authorité d’enregistrement** (RA).
  - **Authorité de dépot** (Repository).
  - **Authorité de recouvrement**.

Composants d’une PKI

- Authorité d’enregistrement (RA) :
  - Vérifie l’identité du demandeur de certificat.
- Authorité de certification (CA) :
  - Signe les certificats.
  - Signe les révocations de certificats.
  - Peut être la même que la RA.
- Autorité de dépot (Repository) :
  - Maintient les certificats dans un *répertoire public* de certificats.
  - Maintient une *liste de révocation de certificats* (*CRL*) dans le répertoire des certificats. La CRL est vérifiée activement par les clients ou par les services de validation.
- Autorité de recouvrement :
  - Protège certaines clés privées pour récupération ultérieure.
- Clients.

Clients et mode d'emploi

- Actions à entreprendre par Alice pour joindre la PKI :
  - Génération de sa paire clé privée/clé publique.
  - Transport de sa clé publique à la RA :
    - **Certificate Signing Request** – divers formes.
  - La RA vérifie qu'Alice est bien celle qui prétend l'être (à définir selon les implémentations !) et informe la CA.
  - La CA délivre le certificat stipulant "Cette clé 'K_A appartient bien à Alice".
  - Le Repository récupère et stocke ce certificat, mis à disposition pour tout le monde.
- Maintenant tout Bob peut récupérer (soit chez le Repository, soit d'une autre manière !) le certificat d'Alice
- Si Bob *fait confiance* à la CA, peut accepter comme valide le certificat, et authentifier diverses conversations avec Alice.
- Celà implique bien sûr la *possesion* de la clé publique de la CA, pour bien vérifier la signature digitale sur le certificat !

CSR en BouncyCastle

- Classe `PKCS10CertificationRequest`.
- Constructeur :
  `PKCS10CertificationRequest(signatureAlgorithm, subject, pubkey, attributes, signingKey)`.
  - `signingKey` est la clé privée correspondant à la clé publique – certificat *auto-signé*.
  - `attributes` peut être `null`.
- Méthode `verify()` – vérifier la requête autosignée.
- Méthode `getPublicKey()` pour retrouver la clé à signer.
- Méthode `getCertificationRequestInfo()` qui renvoie un objet de type `CertificationRequestInfo`.
  - Méthode `getSubject()` de cette classe, permettant de retrouver le propriétaire de la clé publique à signer.
- Méthode `getAlgorithmIdentifier()` pour retrouver l'algorithme pour lequel la clé est utilisable.

Chaînes de confiance

- Deux approches :
  - Construction d’une hiérarchie d’authentification, où la clé publique de la racine serait généralement reconnu.
  - Graphe arbitraire de la relation d’authentification des certificats, en s’appuyant sur la connaissance individuelle des “certificateurs”.
- Modèle de confiance directe : tous les utilisateurs sous-scrivent à la même CA.
  - Tous les certificats pourront être placés dans le même répertoire.
  - Les certificats pourront être retransmis d’utilisateur à utilisateur.

Chaînes de confiance

- Modèle de confiance hiérarchique : pour des grandes communautés d'utilisateurs.
  - Nombre de CA qui fournissent chacune sa propre clé publique à une fraction d'utilisateurs.
  - *Arbre de confiance :* certificats "racine", certificats qui authentifient des certificats qui authentifient...
  - La validité d'un certificat "feuille" est vérifiée en traçant en arrière le chemin de confiance, jusqu'à ce qu'on trouve un certificat d'une authorité à laquelle on fait confiance.
  - Exemple : X.509.

Révocation de clé/certificat

- *Liste de révocation de certificats* (CRL) signée et maintenue par la CA.
- Utilisation :
- Les clients vérifient eux-mêmes activement les CRLs (éventuellement en utilisant une mémoire cache).
- Raisons pour la révocation :
  - La clé privée de l’utilisateur est supposée compromise.
  - L’utilisateur n’est plus certifié par la CA.
  - Le certificat de la CA est supposé compromis.
  - La validité de l’association identité-clé publique n’est pas garantie (erreurs dans la RA).
- Chaque CA maintient une liste de certificats révoqués mais non expirés qu’elle signe.

– Il ne faut pas confondre révocation et date limite !

CRL en Java et BouncyCastle

- Classe `java.security.cert.X509CRL` encapsulant une CRL générée par un serveur de CRL.
  - Méthodes : `verify(PublicKey), getIssuerX500Principal()`.
  - Méthode de recherche `X509CRLEntry getRevokedCertificate(BigInteger serialNumber)` ou `getRevokedCertificate(X509Certificate certificate)`.
  - La classe `X509CRLEntry` contient `getRevocationDate()` et `getSerialNumber()`.
  - La classe `CertificateFactory` permet de récupérer une CRL sur un flux d'entrée à l'aide de la méthode `generateCRL(InputStream)`.

CRL en BouncyCastle

- Classe `org.bouncycastle.x509.X509V2CRLGenerator` permettant de créer des CRL.
  - Constructeur `X509V2CRLGenerator()`.
  - Méthode `addCRLEntry(userCertificate, revocationDate, raison)`, le `userCertificate` est un `BigInteger` qui représente le Serial Number du certificat.
    - Les raisons sont des entiers membres statiques de la classe `CRLReason`, dans l'ordre : `unspecified=0, keyCompromise, cACompromise, affiliationChanged, superseded, cessationOfOperation, certificateHold, removeFromCRL, privilegeWithdrawn, aACompromise`
  - Méthode générant une CRL : `X509CRL generate(java.security.PrivateKey key)`.
  - Autres méthodes : `setIssuerDN(issuer), setThisUpdate(date), setNextUpdate(java.util.Date date), setSignatureAlgorithm(signatureAlgorithm)`.

Vérification en ligne des CRL

- La nécessité de vérifier *en ligne* la validité d'un certificat peut ouvrir la porte à des attaques DoS contre les CRL.
- Une CRL est fournie toute entière à qui le demande – le client doit la parser pour vérifier si un certificat est révoqué.
  - Cela peut poser problème aussi du point de vue de l'intimité !
- Variante : protocole **OCSP**
  - Un serveur OCSP répond à des demandes de vérification de validité de certificats, en renvoyant une confirmation *signée* de l'état d'un certificat à un moment donné.
- Tout système PKI a une *latence* sur l'authenticité :
  - Cela permet d'utiliser des *caches* CRL, ou de réponses OCSP.

OCSP en BouncyCastle

- Classe `OCSPReqGenerator`, constructeur sans paramètres.
  - Insertion de certificats à vérifier : `addRequest(certId)`, argument de type `CertificateID`,
  - Argument constructible avec `CertificateID(CertificateID.HASH_SHA1, issuerCert, serialnumber)`.
  - Définir `setRequestorName(X500Principal requestorName)`.
  - Génération de la requête : `OCSPReq generate()` (non-signée), ou `generate(signingAlgorithm, privatekey, X509Certificate[] chain, provider)` (requête signée).
- Classe `OCSPReq` encapsulant une requête :
  - Méthode `Req[] getRequestList()`, renvoyant un tableau de `Req` dont la méthode `CertificateID getCertID()` permet de retrouver le Serial Number pour chaque certificat composant la requête.
  - Test d'authentification : `boolean verify(publickey, sigProvider)`.

OCSP en BouncyCastle

- Construction d'une réponse à l'aide de la classe `BasicOCSPRespGenerator` :
  - Constructeur `BasicOCSPRespGenerator(publickey)`.
  - Rajout d'une réponse identifiée par le Serial Number : `addResponse(CertificateID certID, CertificateStatus certStatus)`.
  - Interface `CertificateStatus` implémentée dans deux classes : `RevokedStatus, UnknownStatus` et un membre statique, `GOOD`.
  - Constructeurs `RevokedStatus(revocationDate, int reason)` et `UnknownStatus()`.
  - Génération de la réponse : `BasicOCSPResp generate(signingAlgorithm, privatekey, X509Certificate[] chain, Date thisUpdate, provider)`.
- Retrouver les certificats dans une réponse de type `BasicOCSPResp` :
  - `X509Certificate[] getCerts(provider)` – ensuite il faut vérifier que le certificat pour lequel on a demandé la réponse se trouve dans cette liste. mais contenant que des certificats.
  - Vérifier la signature : `boolean verify(publickey, sigProvider)`.
- Nécessaire d'ajouter des nonces dans les requêtes et les vérifier dans les réponses pour éviter des attaques de rejeu.

Réseau de confiance

- ⧫ Inclut les systèmes de confiance directe et hiérarchique.
- ⧫ Se base sur l'idée que plus d'information implique plus de confiance.
- ⧫ Un certificat peut être co-signé par *plusieurs* authentificateurs.
- ⧫ Exemple : *PGP (Pretty Good Privacy* (P. Zimmermann), utilisée pour la confidentialité des e-mails.
  - Utilise une infrastructure de clés publiques basé sur les certificats.
  - Les certificats se basent sur une notion de *confiance* : **zéro**, **partiel** et **complet**.
  - Les différentes signatures pour une seule clé pourraient avoir des niveaux différents de confiance.
  - Les certificats peuvent même être auto-signés.
  - Une signature digitale devient une forme d'*introduction* de l'association identité/clé authentifiée.
  - Plus de chemins de certification, plus de confiance dans la clé.
  - Exemple...

Projet PKI

- *Objectif :* créer un ensemble d'applications client-serveur implémentant une PKI.
- Protocôle d'enregistrement d'un client auprès de sa RA.
  - Protocôle hors-ligne – sinon pbs. d'attaque Man-In-The-Middle, ou nécessité d'utiliser une autre infrastructure de connexion sécurisée.
- Protocôle de demande et d'obtention d'un certificat signé par sa CA.
  - Après la génération d'une nouvelle clé publique client.
- Protocôle de communication entre clients.
  - Peut engendrer une échange de certificats entre interlocuteurs, au démarrage.
  - Ou peut se baser sur un autre protocôle de récupération du certificat de l'interlocuteur, auprès d'un répertoire des certificats.
  - Exemple : utiliser Needham-Schroeder pour la création d'une clé de session.
- Politique de gestion de la CRL :
  - Implémentation des deux méthodes : demande périodique de CRL et vérification d'appartenance dans la CRL et vérification par OCSP lors de chaque nouvelle connexion.
  - Protocole à définir pour demander le rajout d'un certificat dans une CRL.
- Implémentation du protocole Needham-Schroder pour une application de type chat sécurisé, avec une gestion de type PKI des clés et génération de clé de session à partir de la nonce N_B.

Travail demandé

- Travail en binome.
- Fichiers source (et éventuellement un jar) avec votre implémentation.
  - Attention à la portabilité !
- Rapport contenant les points suivants :
  - Description (formalisée comme vu en cours) des protocôles de communication, avec analyse de la sécurité du protocôle (assertions !).
  - Description de l'implémentation de chaque protocôle.
  - Description de l'architecture de votre application (clients/serveurs)
  - Configuration nécessaire et manipulation.
  - Description de votre plan de travail : comment avez-vous conçu les phases d'implémentation, quand avez-vous pensé commencer et quand finir chaque phase, et ensuite comment cela s'est réellement passé.
  - Questions diverses : difficultés particulières rencontrées, aide externe, discussions que vous avez eu avec d'autres binômes ou dans des groupes de discussions, etc.
- *Soutenance* : présentation de votre implémentation.
  - Prévoir une petite application qui montrerait toutes les diverses fonctionnalités implémentés.
- Soutenances : première semaine après les examens.